JOURNAL OF INFORMATION SCIENCE AND ENGINEERING 25, 779-791 (2009)

A Template-based Recognition System for On-line Handwritten Characters*

FEDERICO PRAT, ANDRÉS MARZAL, SERGIO MARTÍN, RAFAEL RAMOS-GARIJO AND MARÍA JOSÉ CASTRO+
Dep. de Llenguatges i Sistemes Informàtics
Universitat Jaume I
Castellón, Spain
+*Dep. de Sistemas Informáticos y Computación*
Universidad Politécnica de Valencia
Valencia, Spain

New developments on our real-time recognition engine for isolated handwritten characters are presented. This engine is based on approximate Dynamic Time Warping comparisons with prototypes selected by fast, less accurate classification procedures. The error rate it currently obtains on the standard Pendigits task, 0.60%, is significantly lower than both the error rate of other recently published techniques and the one we obtain from the recognition engine included in the Microsoft Vista operating system.

Keywords: on-line handwritten text recognition, template-based recognition, dynamic time warping, nearest-neighbours, filter-and-refine classification

1. INTRODUCTION

Pen-based input is experiencing great demand since the advent of pen-based computers and devices, such as Tablet PCs, Personal Digital Assistants, digitizing tablets, and pressure sensitive screens. State-of-the-art techniques, though not perfect, allow the use of on-line character recognition in practical applications. Thus, recent operating systems integrate input controls and text recognition engines to ease programming applications that offer pen-based interaction. For instance, Microsoft XP Tablet PC Edition and Microsoft Vista contain powerful recognition engines which can be easily accessed by the user via an input panel.

We have recently designed and built an open source recognition engine for isolated characters and we have integrated it into our own text input panel, a portable one that mimics Microsoft's widget and is aimed at improving it. Our engine, first presented at CCIA 2007 [1] and in continuous development, offers state-of-the-art performance for isolated character recognition and works in real time. In this paper, we present the recognition engine and its more recent improvements, readjusting its parameters from new experimental results on our own alphanumeric corpus, UJIpenchars [2], publicly available at the UCI Machine Learning Repository [3]. The engine recognition code is available at ftp://acrata.act.uji.es/pub/MIRlibInk.zip.

Our engine uses a 3-nearest-neighbours classifier with an approximate Dynamic Time Warping dissimilarity measure. In order to accelerate the process, two different fast

Received February 1, 2008; accepted November 28, 2008.
Communicated by Yau-Hwang Kuo, Pau-Choo Chung and Jar-Ferr Yang.
* This work partially supported by the Spanish *Ministerio de Educación y Ciencia* (TIN2006-12767 and Consolider Ingenio 2010 CSD2007-00018), the *Generalitat Valenciana* (GV06/302), and *Bancaixa* (P1·1B2006-31).

filtering procedures are applied to the prototypes and only a subset of them is considered by the classifier. On our UJIpenchars standard writer-independent task, this improved engine currently obtains a 10.85% error rate and runs in real time: it classifies more than 50 characters per second on a conventional laptop. On the Pendigits [4] task, it presents an excellent 0.60% error rate, significantly lower than both the error rate of other recently published techniques and the one we obtain from the recognition engine included in the Microsoft Vista operating system.

The remainder of this paper is organized as follows. Next section explains how handwritten characters are represented in order to feed our system. Section 3 describes this recognition engine in detail, along with the empirical work carried out for adjusting its parameters. Our current engine is compared with both its previous version and the Microsoft Tablet PC SDK recognition engine in section 4. Finally, section 5 presents the conclusions.

2. DIGITAL INK AND ITS PREPROCESSING

Digital ink is a time-ordered sequence of strokes and each stroke is a time-ordered sequence of "packets". Each packet contains information about the pen at a given instant: (x, y) coordinates, velocity, pressure, *etc*. We only use (x, y) coordinates, the simplest kind of packets, since we seek portability and coordinates are the least common denominator that pen devices offer. We use the term glyph to name ink representing a symbol.

In our system, each glyph is preprocessed to correct slant and to normalize its dimensions and structure (see Fig. 1). In order to estimate the slant, vectors determined by every pair of consecutive points in the same stroke are considered, but only vectors whose angle difference with respect to the vertical axis is equal or less than 50 degrees are selected. In order to make all these selected vectors point towards a growing y direction, the signs of both vector components are changed if the y one is negative. Then, the angle of the summation of selected vectors is considered to be the glyph slant angle, which is corrected by applying the appropriate shear transform to all glyph points. After slant correction, the glyph is scaled, preserving its aspect ratio, in order to be boxed in a rectangle whose longest side measures one. Then, all coordinates in the glyph are made relative to the center of mass of the points. If the glyph contains two or more strokes, they are concatenated by simply preserving its time-ordered point sequence and discarding its stroke-level structure.

In our system, normalizing steps described above are applied to all glyphs as a common preprocessing. Moreover, when two glyphs are going to be compared by using a certain dissimilarity measure, appropriate glyph versions are employed depending on the kind of measure. Different glyph versions are obtained by applying additional processing steps to the result of the common preprocessing. An important additional processing step in our system, aimed at attaching angular information to merely positional coordinates, is one we refer to as segment-based representation: in order to avoid end-point problems in the definition of angles, each straight segment determined by a pair of consecutive points, p_k and p_{k+1}, is represented by its middle point and the angle of $\overrightarrow{p_k p_{k+1}}$. Thus, the segment-based representation of an n-point glyph version is a sequence of $n - 1$ point/angle pairs.

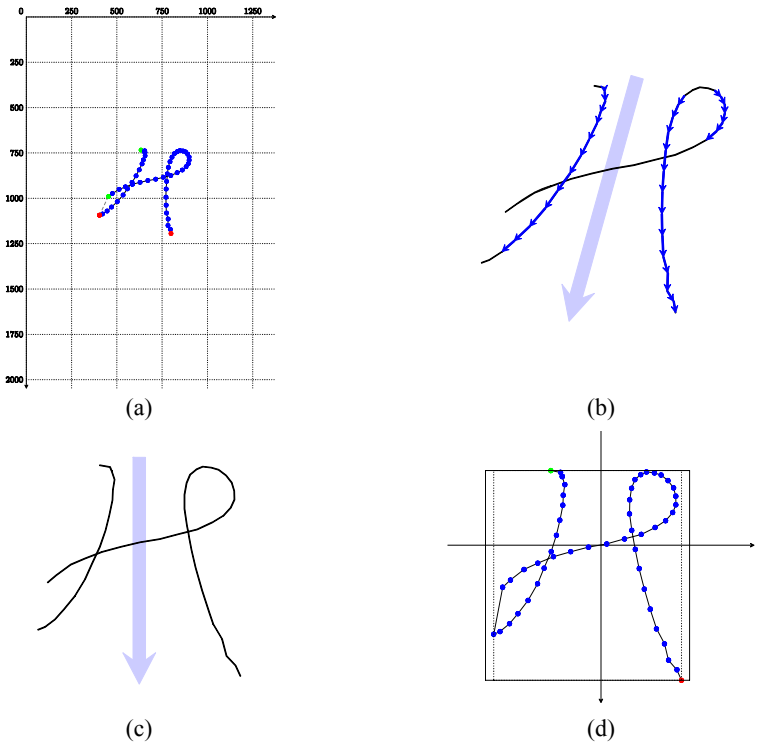


Fig. 1. Common preprocessing; (a) Sample glyph as drawn by the writer; (b) Only pairs of consecutive points whose angle is between certain limits (shown as vectors) are taken into account when computing the estimated slant (shown as an arrow in the background); (c) The slant can be corrected by means of a shear transform; (d) Normalized version of the glyph: the slant is corrected, the scale is normalized to fit into the unit square without affecting the aspect ratio, the origin of coordinates is placed at the center of mass of the points, and the strokes are joined.

3. THE RECOGNITION ENGINE

Our engine has to assign a category label to each glyph it is presented to for recognition, and this classification must be performed in real time, *i.e.* it should take much less than one tenth of a second to provide users with a response. To achieve this, we follow a template-based approach to recognition. So, our system needs a corpus of correctly labelled glyphs, to be referred to as training set, before starting to classify new, test glyphs. Glyphs in the training set are considered prototypes of their categories, and are preprocessed in order to store different versions appropriate for the different dissimilarity measures to be employed for the classification of new glyphs.

As a first approach, we can consider just a dissimilarity measure and, therefore, just one preprocessed version of each prototype stored by our system. A new glyph, after suffering the same preprocessing, could be classified as belonging to the category of its nearest prototype according to the chosen dissimilarity measure.

Thus, it makes sense to first consider a well-known technique for sequence com-

parison such as Dynamic Time Warping (DTW), introduced by Vintsyuk [5] as a comparison method for speech, but also appropriate for digital ink. Unfortunately, DTW is computationally intensive and does not fulfill our real-time response requirement on currently conventional hardware. Sakoe and Chiba [6] proposed a technique that speeds up DTW by sacrificing correctness but, as we will see, it is still not fast enough for our purposes. On the other hand, some simpler comparison techniques are much faster than DTW, but they provide too high error rates. So the approach we finally implemented in our engine is a kind of trade-off where fast techniques are employed to preselect a few prototypes and accelerated DTW takes a classification decision by comparing the test glyph only with prototypes in this reduced set.

In next subsections, we will present the different techniques employed in our engine and how experimental results on the UJIpenchars task [2] guided our decisions on how to use them. The UJIpenchars corpus contains glyphs of the 10 digits, the 26 lowercase letters, and the 26 uppercase letters from 11 writers; each person wrote two instances of each symbol on a Tablet PC, totaling 1,364 glyphs. Some examples from the corpus are illustrated in Fig. 2. Since there are some indistinguishable handwritten characters, such as lower and upper "o" and zero, lower and upper "s", *etc.*, only 35 categories are considered in its standard classification task: 9 for the "1" to "9" digits and 26 for the lower and uppercase versions of each letter, where zero is included in the "o" class. This task is a writer-independent one: all glyphs in the corpus are to be classified by considering as training set only those glyphs provided by the other 10 writers.

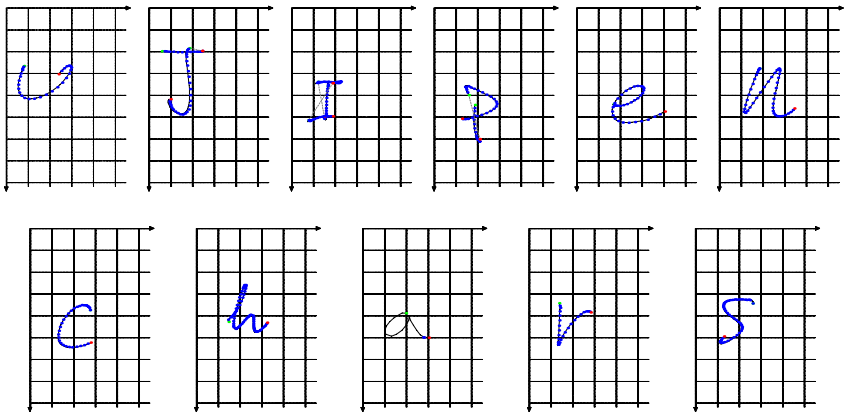


Fig. 2. Some examples from the UJIpenchars corpus.

All the experiments presented in this paper were run on a laptop PC with an Intel Core2 CPU T5600 at 1.83 GHz and 2 Gb of RAM on the .NET platform under the Microsoft Vista Business Edition operating system. The programs were coded in C# without unsafe code.

3.1 Classification with DTW-Based Comparisons

Given two sequences of possibly different lengths, $A = a_1 a_2 \ldots a_m$ and $B = b_1 b_2 \ldots$

b_n, where each a_i and each b_j belong to some set Σ, and a dissimilarity measure δ between elements in Σ, δ: $\Sigma \times \Sigma \to \mathbb{R}$, a very natural way to apply δ to the comparison of sequences A and B consists in averaging local dissimilarities between "corresponding" a_i and b_j elements. DTW provides a well-founded way of determining which element pairs correspond to each other by solving an optimization problem: the minimum-cost path in a directed graph like the one in Fig. 3, from the initial circular node to the top-right corner one. The cost of each arc mainly depends on the node it arrives to: if it is the one at column i and row j, the arc cost will have a factor $\delta(a_i, b_j)$. Moreover, such a cost may be affected by a weight associated to the arc direction: horizontal, vertical, or diagonal. In the most general case, the cost of the optimal path arriving at node (i, j) can be recursively expressed as

$$C(i,j) = \min \begin{Bmatrix} C(i-1,j) + w_h \delta(a_i, b_j), \\ C(i,j-1) + w_v \delta(a_i, b_j), \\ C(i-1,j-1) + w_d \delta(a_i, b_j) \end{Bmatrix}. \tag{1}$$

When weights are set as $w_h = w_v = 1$ and $w_d = 2$, the dissimilarity between sequences A and B can be easily defined in a length-normalized way as

$$D(A, B) = C(m, n)/(m + n). \tag{2}$$

The DTW measure D can be efficiently computed in $O(mn)$ time and $O(\min(m, n))$ space by applying Dynamic Programming to the computation of $C(m, n)$ in the graph of Fig. 3. Moreover, Sakoe and Chiba [6] proposed a technique to speed up DTW computation by sacrificing correctness: the node set involved in the search for the minimum-cost path in the DTW acyclic graph is then limited by a given maximum vertical distance d from the bottom-left-to-top-right diagonal. For instance, Fig. 3 shows as grey nodes those ones that would never be visited for $d = 2$: they would be too far from diagonal nodes, marked with dots. Since we always place the longest sequence on the horizontal axis (in order to ensure that diagonal nodes, one per column, will form a connected path), the running time of this algorithm is $O(\max(m, n)d)$. Smaller values of d result in faster but less accurate DTW estimations.

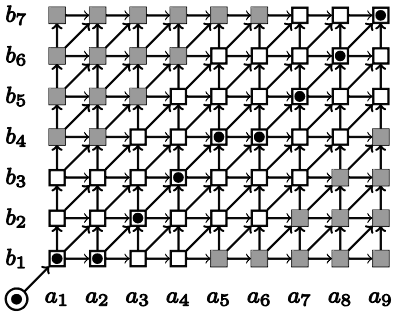
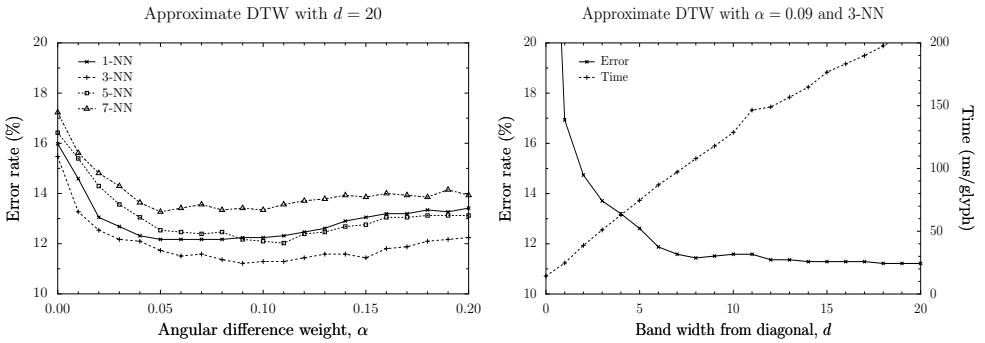


Fig. 3. An acyclic graph for DTW.

In order to completely specify the DTW measure, the local dissimilarity function δ must be defined. We have studied a linear combination of squared Euclidean distance between points and minimum angular difference applied to the point/angle pairs of segment-based representation of glyphs, where the angular difference, measured in radians, is affected by a factor $\alpha \geq 0$.

Finally, classifying a test glyph as belonging to the category of its most similar prototype is just a particular case of the more general k-Nearest-Neighbours (k-NN) rule: each one of the k more similar prototypes votes for its own category, and the winner category becomes the classifier response. Ties can be solved by applying 1-NN to just the prototypes voting for the tied categories.

So there are three parameters to be empirically set: distance d determining the $2d + 1$ width of the Sakoe-Chiba band; the weight α affecting angular distances in the dissimilarity measure δ between point/angle pairs; and the number of prototypes k for the k-NN classification rule. We first considered a large band width by provisionally setting $d = 20$ and measured error rates for several values of α and k (main results are plotted in Fig. 4 (a)). We found that $k = 3$ is the best choice for every value we had considered for α and that the lowest error rate, 11.22%, is achieved at $\alpha = 0.09$. As expected, the time cost, about 200 milliseconds per glyph (ms/glyph), makes the approach unsuitable for real-time recognition. Once k and α were fixed, we tried to determine how much DTW band width can be reduced without affecting classification results too much, and results of the corresponding experiment are plotted in Fig. 4 (b). The choice $d = 18$ preserves the error rate for $d = 20$, but does not reduce time cost significantly. Another interesting value for d is 8: error rate 11.44% is slightly worse, but time cost is reduced to a half. However, to achieve less than 25 ms/glyph, d must be reduced to 1, and then error rate grows to 16.94%. Thus, empirical results showed that real-time recognition requires a different approach.

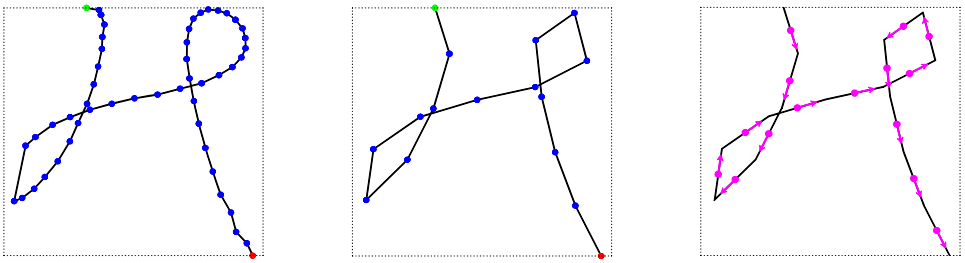


(a) (b)

Fig. 4. Experimental results for (a) setting parameters α and k and (b) reducing parameter d.

3.2 Fast Comparison Techniques

The main reason for DTW to be computationally expensive is its search for a path determining which element pairs correspond to each other in the comparison of two possibly-different-length glyph representations. By using an appropriate resampling as a glyph



(a) (b) (c)

Fig. 5. Segment-based resampling; (a) Normalized glyph after common preprocessing; (b) A fixed number, $m + 1$, of equally-spaced points are sampled along the path; (c) The final result is a sequence of m point/angle pairs.

processing step, glyph versions of fixed length can be obtained for the application of faster dissimilarity measures. The first additional processing step we follow for representing a sequence-of-points glyph resulting from common preprocessing as a sequence of m point/angle pairs consists in resampling the glyph arc at $m + 1$ points equally spaced along their arc length, assuming straight lines between every pair of consecutive points in the original glyph. After this, a segment-based representation (as explained in section 2) provides the desired m point/angle pairs. This combination of two processing steps will be referred to as "resampling to m segments" (see Fig. 5).

3.2.1 One-to-one alignment

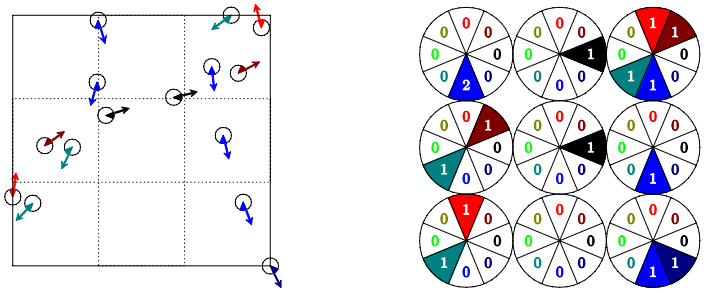
Once glyphs are represented by sequences having the same length, m, we can compare them by means of a one-to-one alignment:

$$D(A,B) = \sum_{1 \le i \le m} \delta(a_i, b_i). \tag{3}$$

This value can be computed in $O(m)$ time, which is significantly faster than DTW techniques. The resampling size m should be carefully chosen, since both the error rate and the running time depend on it. For the parameter α affecting δ, we kept the previously fixed value, 0.09.

3.2.2 Region/direction-histogram comparison

As a second fast comparison technique, we have explored one where glyphs are represented as histograms. After common preprocessing and resampling to m segments, the minimum bounding box of the resulting glyph version is partitioned into 9 regions with a 3×3 grid. Then, for each point/angle pair, the point is replaced with its corresponding region label and the angle is discretized using a standard 8-direction code. Finally, from the resulting sequence of m region/direction pairs, a histogram consisting of $9 \times 8 = 72$ pair counts is obtained (see Fig. 6). If we let a_i and b_i now denote the values at the ith histogram cells for glyphs A and B, the two histogram distance functions we have studied can be expressed as follows:



(a) (b)

Fig. 6. Histogram representation of a glyph; (a) Transformation of its sequence of point/angle pairs into a set of region/direction pairs; the glyph bounding box has been divided into 3 × 3 regions and each angle has been discretized using a standard 8-direction code; (b) The same glyph as a histogram consisting of 3 × 3 × 8 cells.

• the Manhattan distance, as

$$D(A,B) = \sum_{1 \le i \le 72} |a_i - b_i|; \tag{4}$$

• a χ^2-like distance, as

$$D(A,B) = \sum_{\substack{1 \le i \le 72 \\ a_i + b_i > 0}} \frac{(a_i/m - b_i/m)^2}{(a_i + b_i)/2m}. \tag{5}$$

Both distances can be computed in $O(1)$ time, independently of m. Thus, the resampling size m only marginally affects total time classification costs because of its influence in glyph processing steps, but the error rate may strongly depend on it. Moreover, both the error rate and the running time depend on the distance employed for histogram comparison.

3.2.3 Discussion on fast methods

Fig. 7 shows the error rate (left) and the time needed to classify a glyph (right) as a function of the resampling size m. The best 3-NN error rate, 17.08%, is obtained by the χ^2-like distance when m = 130 employing less than 6 ms/glyph. The other histogram distance, the Manhattan one, is faster but more inaccurate: it gets its best error rate, 18.33%, at m = 60 in less than 2.5 ms/glyph. One-to-one alignment, which provides clearly worse results than histogram-based distances, presents its best error rate at m = 90: 19.50% in around 6 ms/glyph; at m = 20, a similar error rate, 19.57%, is achieved with time cost reduced to a half. Thus, these comparison techniques are actually much faster than DTW, but the error rates they provide are too high for such techniques to be employed as final classifiers in a practical recognition engine. However, they can be used to focus the search of a 3-NN DTW-based classifier on a reduced set of prototypes, as we will see in the next subsection.

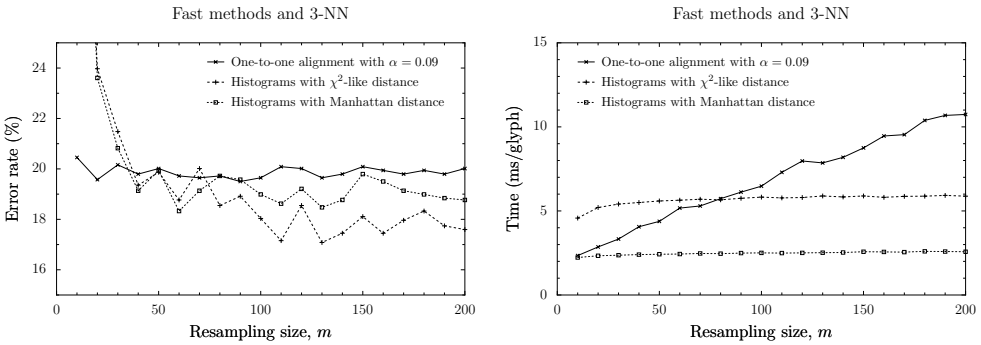


Fig. 7. Experimental results for choosing fast comparison methods and their parameters.

3.3 The Two-Stage Recognition Procedure

In our previous paper [1], fast comparison methods were employed for preselecting categories. Given a fast method, a test glyph, and a given number c of desired candidates, the method ranked prototypes first, then only one best prototype per category was retained, and the c-best categories in the resulting category ranking were returned; finally, all prototypes in the training set belonging to these c candidate categories were included in the selected prototype set to be considered by the approximate DTW classifier. The rationale behind that design choice was to provide the final classifier with all the information that could be useful for choosing between preselected possible responses. This could have, however, a couple of disadvantages. On one hand, prototypes belonging to candidate categories but being very different from the test glyph are not going to influence the classification decision, but their DTW comparisons are going to slow down the system. On the other hand, how parameter c affects classification times is too much task dependent: how many prototypes will be passed to the final classifier per each selected category? Thus, we have recently decided to study a different, more direct way of interpreting the number c of desired candidates: in this case, only the c-best prototypes, as ranked by the corresponding fast method, would be included in the preselected prototype set.

In order to fairly compare different engine settings, error rates were plotted against their time costs, as shown in Fig. 8. Only settings consuming less than 25 ms/glyph were considered. For each comparison method (approximate DTW, one-to-one alignment, and histogram comparison), parameters were set to the values providing best 3-NN error rates in previously shown experiments. Selecting candidates using only one-to-one alignment did not provide interesting results, as expected, and we found out that the best error rate, 10.85%, can be achieved in only 18 ms/glyph when both one-to-one alignment and our χ^2-like distance select 20 candidate prototypes each one. Though this is the setting finally decided for our engine, as depicted in Fig. 9, additional experiments have shown us that further research is needed in order to conclude more general facts about best method combination. For instance, the same error rate, 10.85%, can be achieved with a similar time cost (14 ms/glyph) applying quite different engine settings: when the Manhattan histogram distance (with $m = 60$) selects 5 candidate categories for an approximate DTW classifier with a reduced band width, $d = 8$.

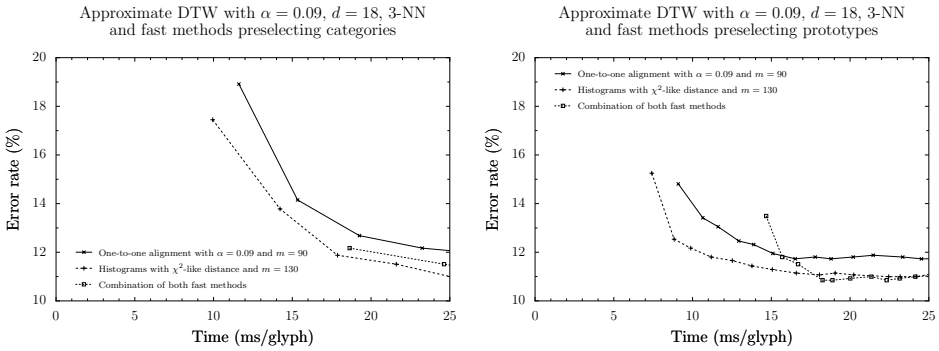


Fig. 8. Experimental results for choosing how to use fast methods in our engine.

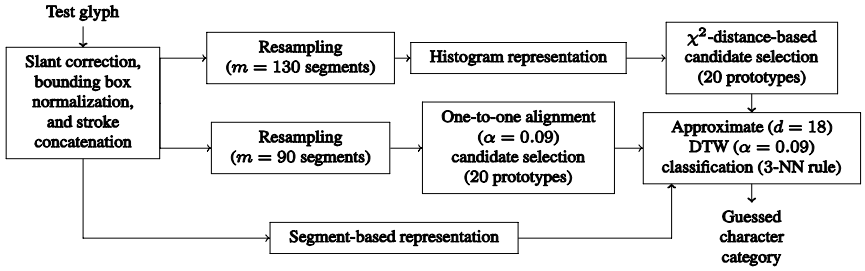


Fig. 9. Recognition engine.

4. COMPARISON WITH OTHER RECOGNITION ENGINES

On UJIpenchars task [2], we can compare our new results with those presented at CCIA 2007 [1]: our old engine achieved an 11.58% error rate in around 20 ms/glyph (and 11.51% in around 30); now we get better recognition results, a 10.85% error rate, without needing more time. We have also run C# experiments with the 1.7 version of the Microsoft Tablet PC SDK recognition engine, using a Microsoft.Ink.RecognizerContext object with an appropriate WordList and flags Coerce and WordMode: it fails the glyph category for 14.74% of the original corpus glyphs (no preprocessing at all) and classification runs in less than 5 ms/glyph. If we help the Microsoft recognizer by providing it with the dimensions of the acquisition box via its Guide property, both error rate and time drop to a very impressive 8.36% in less than 1 ms/glyph, so we plan to study how we can incorporate such kind of positional information in future versions of our engine.

Finally, we have run a standard experiment on the original, non-normalized version of the Pendigits database [4], available at the UCI Machine Learning Repository [3] too. This corpus contains handwritten instances of the 10 digits from several writers: 7,494 glyphs from 30 writers are used as training set and 3,498 glyphs from 14 different writers are used as test data. Our new engine, with exactly the same settings depicted in Fig. 9, obtains on this writer-independent task a 0.60% error rate in 70 ms/glyph. This is an important improvement on our previous result (a 0.83% while tripling classification time) and the error rate is significantly better than both recent results published in the literature

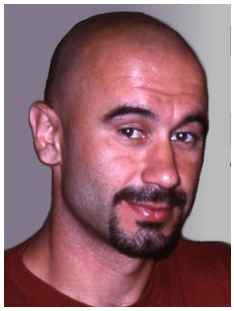
for the same experiment (2.26% in 2005 [7]; 1.66% in 2006 [8]) and the one we get from the Microsoft recognizer even properly setting its Guide property (1.89%; and 4.20% without that setting). However, our engine needs new improvements for keeping recognition times low, as the ones it gets on the UJIpenchars corpus, when facing tasks with larger training sets, as Pendigits is: maybe some kind of test-glyph-independent preselection of prototypes to control the number of fast comparisons our current engine has to make in its first stage.

5. CONCLUSIONS

We have improved our recognition engine for isolated handwritten characters and presented new experimental work for tuning its parameters and for comparing its performance with that of the Microsoft recognition engine. The error rate achieved on a standard experiment with the Pendigits database, 0.60%, is significantly lower than both the error rate of other recently published techniques and the one we obtain from Microsoft's engine. Moreover, we have identified two promising sources for future improvement: using information about glyph position relative to its acquisition box and condensing the training set for retaining just its most informative prototypes.

REFERENCES

1. R. Ramos-Garijo, S. Martín, A. Marzal, F. Prat, J. M. Vilar, and D. Llorens, "An input panel and recognition engine for on-line handwritten text recognition," in C. Angulo and L. Godo, eds., *Artificial Intelligence Research and Development*, IOS Press, 2007, pp. 223-232.
2. D. Llorens, F. Prat, A. Marzal, and J. M. Vilar, "UJIpenchars: A pen-based classification task for isolated handwritten characters," available as UJI Pen Characters data set at [3].
3. A. Asuncion and D. J. Newman, "UCI machine learning repository," http://www.ics.edu/~mlearn/MLRepository.html.
4. E. Alpaydın and F. Alimoğlu, "Pen-based recognition of handwritten digits," available at [3].
5. T. K. Vintsyuk, "Speech discrimination by dynamic programming," *Cybernetics*, Vol. 4, 1968, pp. 52-57.
6. H. Sakoe and S. Chiba, "Dynamic programming algorithm optimization for spoken word recognition," *IEEE Transactions on Acoustics*, *Speech*, *and Signal Processing*, Vol. 26, 1978, pp. 43-49.
7. J. Zhang and S. Z. Li, "Adaptive nonlinear auto-associative modeling through manifold learning," *Lecture Notes in Computer Science*, Vol. 3518, 2005, pp. 599-604.
8. B. Spillmann, M. Neuhaus, H. Bunke, E. Pękalska, and R. P. W. Duin, "Transforming strings to vector spaces using prototype selection," *Lecture Notes in Computer Science*, Vol. 4109, 2006, pp. 287-296.



Federico Prat received his M.S. and Ph.D. degrees in Computer Science from the Universidad Politécnica de Valencia, Spain, in 1991 and 1998, respectively. He currently works as an Associate Professor at the Departament de Llenguatges i Sistemes Informàtics of the Universitat Jaume I at Castellón, Spain. His research interests include pattern recognition, machine learning, and language modeling.

Andrés Marzal received his M.S. and Ph.D. degrees in Computer Science from the Universidad Politécnica de Valencia, Spain, in 1990 and 1994, respectively. He currently works as an Associate Professor at the Departament de Llenguatges i Sistemes Informàtics of the Universitat Jaume I at Castellón, Spain. His research interests include pattern recognition, machine learning, and multimodal interaction.

Sergio Martín received in 2007 his M.S. degree in Computer Science from the Universitat Jaume I at Castellón, Spain, where he currently studies a Master degree in Intelligent Systems. His research interests include image processing, handwritten text recognition and predictive analytics.

Rafael Ramos-Garijo received in 2004 his M.S. degree in Computer Science from the Universitat Jaume I at Castellón, Spain, where he currently studies a Master degree in Intelligent Systems. His research interests include pattern recognition, case-based reasoning and image processing.

María José Castro is currently an Associate Professor of the Departamento de Sistemas Informáticos y Computación at the Universidad Politécnica de Valencia, Spain, where she has taught since 1993. She received her Ph.D. degree in Computer Science from this same University, in 1998. Her research interests include machine learning, speech and handwritten text recognition and language technologies.